Design of a stateless low-latency router architecture for green software-defined networking

Silvia Saldaña Cercós[a], Ramon M. Ramos.[b], Ana C. Ewald Eller[b], Magnos Martinello[b], Moisés R.N. Ribeiro[b], Anna Manolova Fagertun[a], and Idelfonso Tafur Monroy[a]

[a] DTU Fotonik, Technical University of Denmark, Kgs. Lyngby, Denmark;
[b] LabNERDS-Informatics and Electrical Engineering Department, Federal University of Espírito Santo (UFES), Vitoria E.S. Brazil

ABSTRACT

Expanding software defined networking (SDN) to transport networks requires new strategies to deal with the large number of flows that future core networks will have to face. New south-bound protocols within SDN have been proposed to benefit from having control plane detached from the data plane offering a cost- and energy-efficient forwarding engine. This paper presents an overview of a new approach named KeyFlow to simultaneously reduce latency, jitter, and power consumption in core network nodes. Results on an emulation platform indicate that round trip time (RTT) can be reduced above 50% compared to the reference protocol OpenFlow, specially when flow tables are densely populated. Jitter reduction has been demonstrated experimentally on a NetFPGA-based platform, and 57.3% power consumption reduction has been achieved.

Keywords: Energy-efficiency, optical networks, OpenFlow, NetFPGA -

1. INTRODUCTION

Current core network design strategies consolidate towards a cost-efficient solution by implementing a reduced number of core network nodes that support higher bit rates.[1] These strategies, combined with current Internet growth, lead to a key challenge for network operators, namely to cater to the large volumes of traffic. One solution for core networks is to define a forwarding plane of core nodes which ensures intra-domain routing packet transport while mitigating the complexity of network forward engines. Besides low-complexity for the elements in the forwarding plane, flexibility to cope with traffic changes on-demand is also required for meeting future requirements as they arise. Software defined networking (SDN) is a promising technology that offers this flexible and programmable networking platform with the control plane detached from the data plane.[2] Current SDN implementations focus on Ethernet switching primarily for data centers, that can benefit from centralizing the management intelligence.[3] However, some challenges arise when extending SDN for transport network architectures[4] , where scalability is on a major concern.[5] As the network scales up, the number of states related to the active flows in the network increases as well as the number of lookups table operations. This leads to an end-to-end higher latency.[6]

OpenFlow is the most used south-bound protocol within the SDN control architecture. However, it does not offer a simple design for forwarding engines, and it contributes to scalability problems due to the number of states related to the active flows. In this paper a protocol within the SDN architecture named KeyFlow for design of stateless low-latency router architecture is presented. KeyFlows aims at coping with latency challenges that arise in SDN transport network architectures since it does not require states associated to the active flows. KeyFlow reduces the complexity of the core forward engines by replacing the traditional table lookup by a simple operation: a division.[7,8] Consequently, latency linked to the table lookup bottleneck in the data plane of the SDN core networks is eliminated unlocking the potential usage of SDN technology in core networks. Besides latency, power consumption has been a concern in core networks design[9] since network equipment power consumption accounts for 14% of the overall information and communications technology (ICT) sector.[10] Ternary addressable content memories (TCAMs) have become an indispensable resource to provide the throughput and high packet

Further author information: (Silvia Saldaña Cercós E-mail: ssce@fotonik.dtu.dk)

Optical Metro Networks and Short-Haul Systems VII, edited by Atul K. Srivastava, Benjamin B. Dingel, Achyut K. Dutta,
Proc. of SPIE Vol. 9388, 93880I · © 2015 SPIE · CCC code: 0277-786X/15/$18 · doi: 10.1117/12.2077560

processing rates that SDN requires. TCAMs are needed to perform the wildcard matching in flow table lookups despite that they are pricey and power-demanding.[11] Improvements upon the OpenFlow implementation in terms of TCAMs utilization have been presented in literature,[12] where per-port circuitry predicts the flow membership of a packet in order to bypass TCAM lookups. KeyFlow expands the state-of-the-art beyond reported research by investigating the potential benefits for a stateless low-latency and power-efficient router architecture.

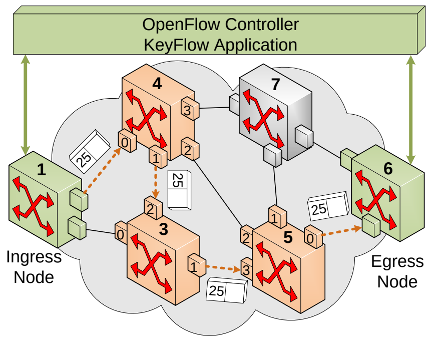
The rest of the paper is structured as follows. First, the operating principle for KeyFlow is detailed. Then, the design and implementation of a KeyFlow switch on a full line-rate 4-port 1 GbE NetFPGA-based is described. Third, latency and jitter results based on an open source platform, Mininet, for a KeyFlow prototype and OpenFlow switch are presented. Then an analysis on power consumption for a reference OpenFlow 1.0 switch is also presented. A comparison between OpenFlow 1.0 and a novel KeyFlow switch from power consumption point of view is performed. Finally, conclusions and future work is presented.

2. KEYFLOW: OPERATING PRINCIPLE

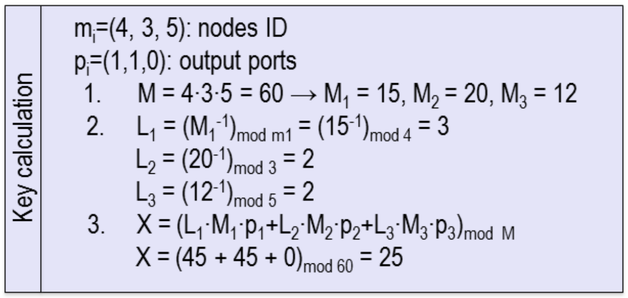
KeyFlow offers the possibility to reduce core network latency by removing the flow table lookup. Instead of the traditional flow table, KeyFlow is based on a simple division which concurrently reduces core network routers' complexity. The forwarding mechanism is based on the well-known residue number system (RNS).[13] The fundamental concept of RNS relies on the idea that every large number can be represented by a combination of small numbers obtained from the reminder of the division of this number by a set of co-primes. KeyFlow uses this mathematical property to define the path a packet follows. The large number represents the path ID, the set of co-primes represent the nodes' ID, and the reminder states the output port for each router.

Fig. 1 illustrates a KeyFlow network architecture example. A flow goes from the ingress node, $m_i = 1$, to the egress node, $m_i = 6$, through the intermediate nodes 4, 3, and 5, $m_i = (4, 3, 5)$. The path is illustrated with dash orange arrows. As depicted, the output ports are $p_i = (1, 1, 0)$ for each of the nodes, respectively. One of the benefits from KeyFlow is that the number of interactions between core nodes and controller is reduced. Communication occurs between node 1 and controller whenever there is no matching rule for a specific packet. The ingress node sends a request to the controller which computes a unique path ID based on the route that the packets needs to follow to reach its destination (e.g., the egress node with ID 6). In the illustrative example from Fig. 1a) this path ID is 25. This path ID is the so-called key. The controller sends a rule to the edge nodes with this information, so that every packet with the same destination is assigned with the same key. The ingress edge node assigns the key onto the packet header. KeyFlow core nodes do not have a flow table. Unlike traditional core intermediate nodes, KeyFlow core nodes perform a simple mathematical operation based on the key in the header of the packet and the node ID to determine the output port. The output port is determined by the reminder of the division between the key and the node ID.

Let $< X >_i$ be defined as the remainder from X/i. In the KeyFlow core node, X is the key at the packet's header and i is its own ID. In the example presented in Fig. 1a) node 4 performs the following operation to



(a)


Key calculation

m_i=(4, 3, 5): nodes ID

p_i=(1,1,0): output ports

1. $M = 4 \cdot 3 \cdot 5 = 60 \longrightarrow M_1 = 15, M_2 = 20, M_3 = 12$
2. $L_1 = (M_1^{-1})_{\text{mod } m1} = (15^{-1})_{\text{mod } 4} = 3$

   $L_2 = (20^{-1})_{\text{mod } 3} = 2$

   $L_3 = (12^{-1})_{\text{mod } 5} = 2$
3. $X = (L_1 \cdot M_1 \cdot p_1 + L_2 \cdot M_2 \cdot p_2 + L_3 \cdot M_3 \cdot p_3)_{\text{mod } M}$

   $X = (45 + 45 + 0)_{\text{mod } 60} = 25$

(b)

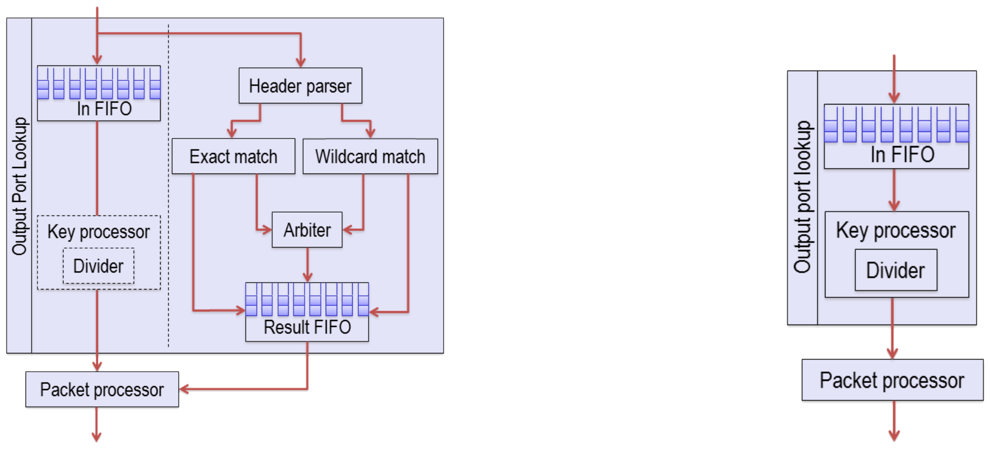
Figure 1: KeyFlow: Operating principle. 1a) KeyFlow network architecture example, and 1b) KeyFlow key calculation example

calculate the output port: $< 25 >_4 = 1$. Analogously, node 3 and 5 perform $< 25 >_3 = 1$, and $< 25 >_5 = 0$, respectively, to calculate their respective output ports. Fig. 1b) shows an example for the key calculation assessed in the control plane assuming that $m_i = (4, 3, 5)$ are the nodes IDs, 25 is the key, and $p_i = (1, 1, 0)$ are the respective output ports. Key calculation consists of three steps. 1) The parameter M_i is obtained by dividing M by each node ID, where M is the product of all nodes IDs. Thus, $M_i = M/m_i$. 2) The parameter L_i is calculated. L_i is the multiplicative inverse of M_i (i.e. $M_i \cdot L_i \equiv 1 \ (mod\ m_i)$). 3) The key, X, is calculated by summing up the product of each L_i, M_i, and output port for each node, $L_i \cdot M_i \cdot p_i \ (mod\ M)$.[7]

3. KEYFLOW AND OPENFLOW IMPLEMENTATION

In this work, the OpenFlow and the KeyFlow design are implemented using NetFPGA as the implementation platform. The hardware design is based on the reference implementations from Stanford University.[14] The 4-port 1 BgE NetFPGA Xilinx Virtex-II Pro 50 has been selected based on the following requirements: a platform capable of supporting high data rates (near-application-specific integrated circuit (ASIC) performance);[11,15] flexible enough to allow the implementation of a completely new core network forward engine. Due to the NetFPGA platform capabilities, it has been used previously to implement an IP-less forwarding fabric named Bloom-filter based.[16]

The OpenFlow reference design implemented on the NetFPGA includes MAC/CPU queues, output port lookup, and user data path modules.[14] From the OpenFlow modular NetFPGA pipeline structure presented in the literature[14] the work presented in this paper is based exclusively on the optimization of forwarding plane. Consequently, only the module associated to output port lookup is detailed. However, latency and power consumption calculations take into account all the modules of the reference pipeline.[14] Fig. 2 presents the modular NetFPGA structure for 2a) a Hybrid switch which includes both KeyFlow and OpenFlow capabilities, and 2b) a KeyFlow switch. For a conventional OpenFlow core network node, when a packet reaches the node, the packet enters into the output port lookup in order to find which output port to forward it to. In the output port lookup module, the header parser extracts relevant information from the packet header. This relevant information is sent to the exact match and the wildcard match in order to find a flow that matches the required information and determines the output port. The exact match uses two hash functions to match an exact flow. This process is performed off-chip on the static random access memories (SRAMs). The wildcard match, on the other hand, uses the TCAMs built using the Xilinx SRL16e primitives.[14] The arbiter module determines which solution to use in case of getting a match from both modules: exact match has higher priority than wildcard match. The forwarding information or associated action selected by the arbiter is stored in the Result first in first out (FIFO) buffer and sent to the packet processor. Fig. 2b) illustrates how complexity is reduced when using a KeyFlow switch as an alternative to the reference OpenFlow switch. The entire lookup process is replaced



(a) (b)

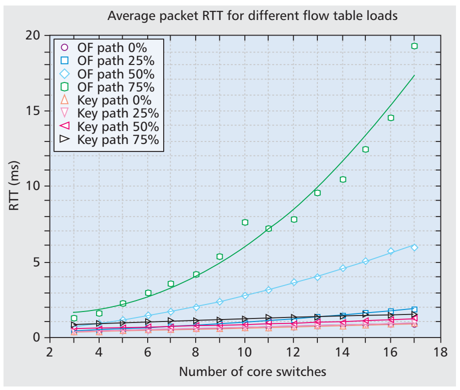
Figure 2: Modular NetFPGA pipeline structure for 2a) a Hybrid switch and 2b) a KeyFlow switch

by a single module which performs the division explained previously. By adding this module and replacing the traditional table look up three benefits are achieved: 1) latency is reduced. An OpenFlow switch by contrast offers a latency which is dependent on the flow table size, 2) jitter is also reduced. Under OpenFlow there is jutter because the delay associated with the table look up depends on where on the list the flow is placed, and 3) the need of using the pricey and power-demanding TCAMs is eliminated, since there is no need of performing wildcard match any longer, and instead a simple operation is done. In order to facilitate the migration towards KeyFlow core networks, a Hybrid switch has been implemented. A Hybrid switch keeps OpenFlow capabilities, but it has the possibility to enable the key processor module presented in Fig. 2a) so that the packet does not go through the header parser and subsequent modules. However, maintaining the modules gives the possibility to use the network fabric as OpenFlow for user convenience.

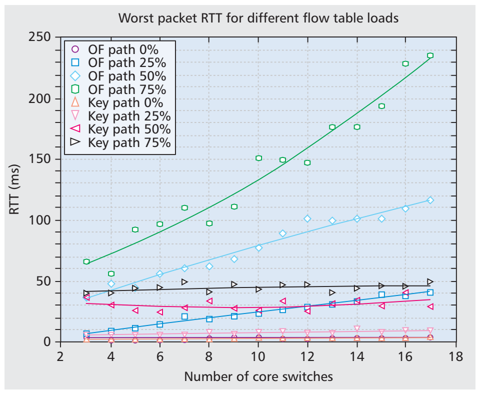
4. REDUCTION OF NETWORK LATENCY

The scope of the presented latency analysis is to determine the switch latency associated with the table lookup bottleneck. This study gives an overview on the end-to-end latency impact when removing table look and replacing it with a simple mathematical operation. Tests have been done on a linear topology assuming that all the forwarding engines are under the same table utilization conditions, for table flow utilization of 0, 25, 50, and 75%. For the latency analysis the emulation platform Mininet has been used as an experimental testbed. For the experiment 5000 64-byte packets are generated and transmitted at constant time intervals. Fig. 3a) and Fig. 3b) show the average and the worse round-trip-time (RTT) for the packets for different flow table utilization as a function of the number of hops, in an emulated network with a reference OpenFlow and a KeyFlow switch. Results show that for a low flow table utilization (i.e., 25%) there is no significant difference on the packet RTT between the KeyFlow and OpenFlow switch. This is because the time needed to perform the division is equivalent to the time needed to execute the frequent cache hits for the lookup process. As the table utilization increases, the KeyFlow switch outperforms the OpenFlow switch. The OpenFlow switch offers an increased packet RTT with the number of hops, whereas the KeyFlow switch presents average values below 3 ms. There is a slight increase on the packet RTT when using a KeyFlow switch. This is expected, since higher number of flows requires additional lookup processing at the edge nodes. For the worse case scenario presented in Fig. 3b), values are one order of magnitude higher than for the average results presented in Fig. 3a). However, similar qualitative performance is observed in both scenarios.

In order to analyze the jitter impact for both an OpenFlow and a KeyFlow switch, the packet RTT has been measured on a NetFPGA-based platform for a single hop and 5000 packets for a) a KeyFlow switch (which does not have a flow table), b) an OpenFlow switch with the flow table utilization set at 0%, and c) the same OpenFlow switch with the flow table utilization set to 100%. As presented in Fig. 3 no differences in terms of

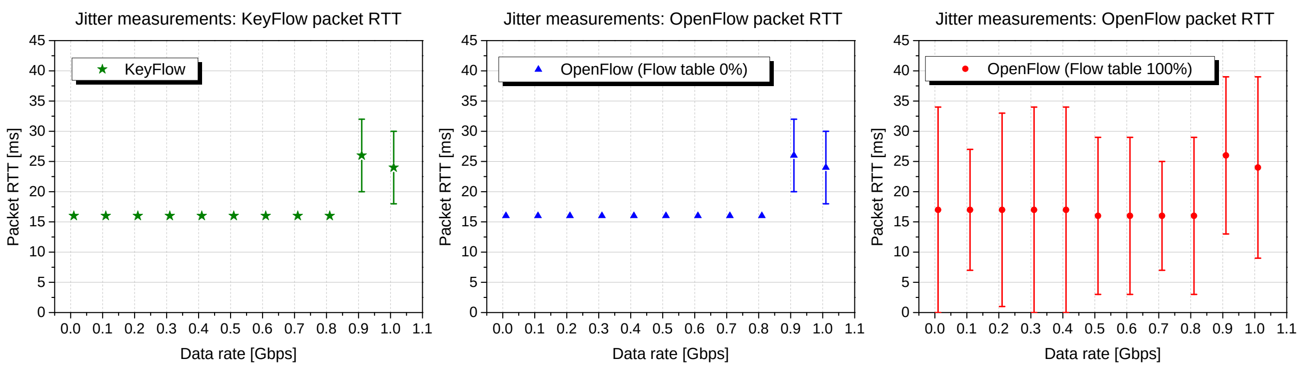


(a)



(b)

Figure 3: RTT comparison between a reference OpenFlow and a KeyFlow switch: 3a) average and 3b) maximum packet RTT



(a) (b) (c)

Figure 4: Jitter measurements based on the packet RTT for 4a) a KeyFlow switch, 4b) an OpenFlow switch with the flow table utilization at 0%, and 4c) an OpenFlow switch with the flow table utilization at 100%

latency are expected for a single hop. Fig. 4a) and Fig. 4b) are in accordance with previous results, showing identical results for both the KeyFlow and the OpenFlow switch with an empty table flow. However, as the flow table utilization increases (Fig. 4), variations on the packet RTT are observed. This error bars represent the jitter impact on an OpenFlow switch since the packet RTT depends on where the rule associated to a specific packet is located on the table. Higher latency and jitter is achieved for bit rates close to the port line rate.

5. CORE FABRIC POWER CONSUMPTION

In this section power consumption measurements are presented. First, the used methodology is described. Then, a set of results is detailed including power consumption of a reference OpenFlow switch and a stateless KeyFlow switch. Power consumption is broken down into different components: 1) Total power consumption is presented accounting for both static and dynamic contributions, 2) Module-based power consumption is described including all the modules defined in the implementation section, and 3) A resource-based overview is given, where the power consumption of each of the NetFPGA resources is stated.

Measuring the power consumption of the two different switches is achieved by following three steps. The presented methodology is associated to the NetFPGA Xilinx platform. The first step includes a gate-level simulator, ModelSim. ModelSim is a Perl-based testing infrastructure which provides functional verification of the design. The outcome of the simulations includes the activity rates for each of the NetFPGA resources. These activity rates, also known as toggle rates, are needed to define how often there is a gate transition, and thus contributing into the overall power consumption. NetFPGA resources constitutes logic resources, signals, input/outputs (IOs), the input clocks for the block random access memories (BRAMs), and the digital clock managers. The simulation tool also provides the frequency for each of the internal clocks. Once these toggle rates are calculated they are exported into a .vsd file. Additionally, the design needs to be synthesized using a Xilinx ISE Virtex II 50 Pro NetFPGA. The synthesized design is described in a native circuit description (NCD) file. This file contains the logic of the design mapped to components. This file is also used to create a physical constraints file (PCF) which is important when determining the overall power consumption.[17] This synthesized design is used in the final step, together with the .vsd file to calculate the power consumption. The Xilinx XPower Analyzer (XPA) tool from Xilinx ISE design software is used. This tool provides the power consumption for each of the components based on the NCD and the toggle rates of each of the NetFPGA resources.

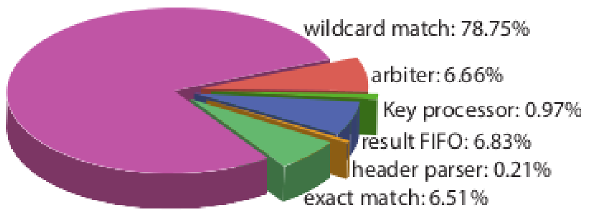
All the results presented in this section have been taken generating a single 1500-byte packet in order to avoid bottlenecks from buffering. Additionally, for the reference OpenFlow switch the flow table is set to 40 flows based on previous work presented in literature.[7]

Table 1 presents the static and the dynamic components of the power consumption for a) OpenFlow switch, b) Hybrid switch, and c) KeyFlow switch. The static power consumption is constant for the three implementations and it is measured to be 159 mW. From the dynamic power consumption perspective, 6% savings and 53.7%

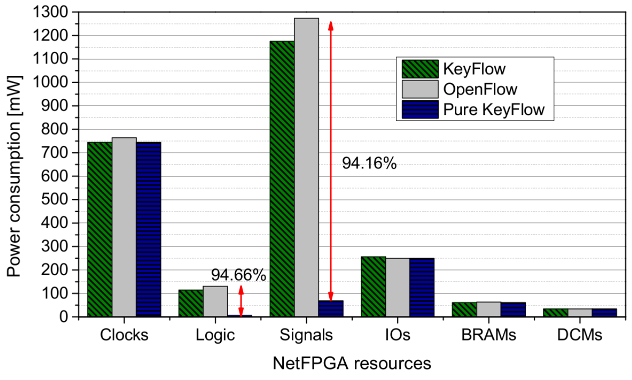
Table 1: Total power consumption [mW]

	Static	Dynamic	Total
OpenFlow	159	2514	2673
Hybrid	159	2384	2543
KeyFlow	159	1164	1323

savings are achieved by the Hybrid and the standalone KeyFlow switch, respectively, compared to the reference OpenFlow switch. This outcome is achieved since the KeyFlow switch does not require the use of TCAMs when performing the table lookup. These results are verified when looking in detail at the contribution of each of the modules for the Hybrid switch. Fig. 5a) depicts a broken down analysis of each of the implementation modules. As expected, the most power-demanding module is the wildcard match since it requires TCAMs. It accounts for 78.75% of the overall power consumption of the output port lookup module. Exact match, result FIFO, and arbiter, which are used in the lookup process contribute with 6.51%, 6.83%, and 6.66%, respectively. The key processor accounts only for 0.97% of the overall power consumption. The data and control signals are the most power demanding resources for the NetFPGA as presented in Fig. 5b). Further analysis on the signals can be achieved by using a NetFPGA analyzer module which captures and allows designers to further work with $control$, $data$, in_{wire}, and out_{ready} signals.[18] A pure KeyFlow implementation, where the entire lookup process is eliminated as presented in Fig. 2b), achieves further savings compared to a KeyFlow switch that mantains the OpenFlow switch capabilities. This outcome is reflected in the resource-based power consumption analysis presented on Fig. 5b). Signals and logic power consumption can be reduced by 94.66% and 94.16%, respectively, compared to the reference OpenFlow switch. The reduction of the complexity of the forwarding engine leads to an overall power savings of 53.7%.



(a)



(b)

Figure 5: 5a) module-based and 5b) resource-based power consumption contributions for an OpenFlow, KeyFlow (or Hybrid), and Pure KeyFlow switch

6. CONCLUSIONS

This article presents a new approach named KeyFlow for a south-bound protocol for SDN in order to offer a cost- and energy-efficient solution for the forwarding plane in transport networks. The proposal is based on an RNS. The presented results include RTT analysis evaluated on an emulation scenario based on a Mininet platform, a jitter comparison between an OpenFlow and a KeyFlow switch implemented on a NetFPGA-based platform, and power consumption comprehensive analysis for a reference OpenFlow switch, a KeyFlow switch, and a hybrid switch that offers both OpenFlow and KeyFlow capabilities. The RTT depends on the number of hops and how much populated the flow tables are. For flow table utilization below 25% no significant difference on the packet RTT is observed between the KeyFlow and the OpenFlow switch. For packets below 3 hops both switches perform similarly. As the number of hops increases KeyFlow outperforms OpenFlow specially for densely populated flow table switches, achieving more than 50% RTT reduction when using the novel KeyFlow approach. KeyFlow is an alternative for current power-demanding high-performance switches. The major power consumption contribution in core network nodes are the TCAMs used for the wildcard match in the table lookup process. Wildcard match accounts for more than 78% of the output port lookup module, and thus, 57.3% power savings are achieved when replacing the traditional table lookup by a simple mathematical operation.

Future work includes embedded KeyFlow on wireless access points for mesh networks, and an enhanced version of KeyFlow to achieve higher power savings tackling the clock frequencies which are the second major power consumption contributor of a forwarding network node. On-chip solutions are the next step towards power-aware low latency SDN transport networks.

REFERENCES

[1] Betker, A., Gamrath, I., Kosiankowski, D., Lange, C., Lehmann, H., Pfeuffer, F., Simon, F., and Werner, A., "Comprehensive topology and traffic model of a nationwide telecommunication network," *Journal of Optical Communications and Networking* **6**, 1038–1047 (November 2014).

[2] Iovanna, P., Michele, F. D., Palacios, J. P. F., and López, V., "E2E traffic engineering routing for transport SDN," *Proc. OFC W1K.3* (March 2014).

[3] Marconett, D., Liu, L., and Yoo, S. J. B., "Optical FlowBroker: load-balancing in software-defined multi-domain optical networks," *Proc. OFC W2A.44* (March 2014).

[4] Gringeri, S., Bitar, N., and Xia, T. J., "Extending software defined network principles to include optical transport," *IEEE Communications Magazine* **51**, 32–40 (March 2013).

[5] Yeganeh, S. H., Tootoonchian, A., and Ganjali, Y., "On scalability of software-defined networking," *IEEE Communications Magazine* **51**, 136–141 (February 2013).

[6] Vencioneck, R. D., Vassoler, G., Martinello, M., Ribeiro, M. R., and Marcondes, C., "FlexForward: Enabling an SDN manageable forwarding engine in open vswitch," *10th CNSM and Workshop* (2014).

[7] Martinello, M., Ribeiro, M. R. N., de Oliveira, R. E. Z., and de Angelis Vitoi, R., "KeyFlow: A prototype for evolving SDN toward core network fabric," *IEEE Network* **28**, 12–19 (March 2014).

[8] Saldaña, S., Oliveira, R. E., Vitoi, R., Martinello, M., Ribeiro, M. R. N., Fagertun, A. M., and Monroy, I. T., "Tackling opeflow power hog in core networks with keyflow," *Electronic letters* (2014).

[9] Saldaña, S., Resendo, L., Ribeiro, M. R. N., Fagertun, A. M., and Monroy, I. T., "Power-aware multi-layer translucent network," *Proc. OFC W2A.53* (March 2014).

[10] Vishmanath, A., Sivaraman, V., Zhao, Z., Russell, C., and Thottan, M., "Adapting router buffers for energy efficiency," *CoNEXT (ACM)* , 19 (March 2011).

[11] Zerbini, C. A. and Finochietto, J. M., "Performance evaluation of packet classification on FPGA-based TCAM emulation architectures," *Globecom* (2012).

[12] Congdon, P. T., Mohapartra, P., Farrens, M., and Akella, V., "Simultaneously reducing latency and power consumption in OpenFlow switches," *IEEE ACM transactions on networks* **22** (June 2014).

[13] Garner, H. L., "The residue number system," *Trans. Electronic Computers* , 140–147 (1959).

[14] Naous, J., Erickson, D., Covington, G. A., Appenzeller, G., and McKeown, N., "Implementing an OpenFlow switch on the NetFPGA platform," *Proc ANCS* (2008).

[15] Jiang, W., "Scalable ternary content addressable memory implementation using FPGAs," *ANCS* (2013).

[16] Keinanen, J., Jokela, P., and Slavov, K., "Implementing zFilter based forwarding node on a NetFPGA," *Proc. of NetFPGA Developers Workshop* (2009).

[17] Xilinx, I., "FPGA editor guide," $http://www.xilinx.com/support/sw_manuals/2_1i/download/fpedit.pdf$ (November 2014).

[18] Goodney, A., Shailesh, N., Mengchen, W., Peigen, S., Vivek, B., and Cho, Y. H., "NetFPGA logic analyzer," *2nd North American NetFPGA developers workshop* (2010).